Claims

[c1] A data compression method comprising the steps of:

(a) reading of input symbols, consider two adjacent symbols forming a pair, counting how many times each pair exists, a pair of two symbols is considered replaceable if both of its symbol positions are replaceable, in this step the number of replaceable occurrences is set to be the count how many times a pair exists

(b) inserting into a priority queue the records that contain the number of replaceable occurrences of pairs

(c) finding from the priority queue the pair having the greatest number of replaceable occurrences

(d) marking the symbol positions of the first occurrence of this pair in question not replaceable, if this marking causes a pair whose one symbol is the marked symbol to lose one replaceable occurrence then immediately changing the number in the pair's record and then immediately moving the pair's record into proper position in the priority queue

(e) studying the other occurrences of this pair and replacing each one of those occurrences whose both symbol positions are replaceable with one new symbol, wherein:

> when a pair's number of replaceable occurrences is about to change then immediately changing the number in the record and then immediately moving the pair's record into proper position in the priority queue

(f) repeating the steps (c)-(e) until the greatest number of replaceable occurrences descents to a predetermined level

(g) if this predetermined level is 1 then traversing the symbol sequence once and if a pair has a second occurrence whose both symbols are positive then replacing this second occurrence with one new symbol

(h) transforming the new symbols into values indicating the position of the first symbol of the first occurrence of that pair the symbol was used to replace added by a constant value greater than any input symbol.

[c2] A method as in claim 1 wherein the step (d) and (e) are as follows:
(d) marking the symbol positions of the first occurrence of this pair in question not replaceable, if this marking causes a pair whose one symbol is the marked symbol to lose one replaceable occurrence then immediately changing the number in the pair's record and then inserting the changed record's index or pointer to the record into an array or list if the index or pointer is not already there
(e) studying the other occurrences of this pair and replacing each one of those occurrences whose both symbol positions are replaceable with one new symbol, wherein:

when a pair's number of replaceable occurrences is about to change then immediately changing the number in the record and then inserting the changed record's index or pointer to the record into an array or list if the index or pointer is not already there,
when all the occurrences of a pair have been studied then this array or list is traversed and the records are inserted into proper positions in the priority queue and the array or list is emptied.

[c3] A method as in claim 1 wherein the step (d) and (e) are as follows:
(d) marking the symbol positions of the first occurrence of this pair in question not replaceable, if this marking causes a pair whose one symbol is the marked symbol to lose one replaceable occurrence then updating the corresponding variable assigned for the case
(e) studying the other occurrences of this pair and replacing each one of those occurrences whose both symbol positions are replaceable with one new symbol, wherein

when a pair's number of replaceable occurrences is about to change then two arrays are used to store the number change, one for the left side changes caused by the disappearing occurrence and one for the right side changes,
these two array also store the needed information for new pairs i.e. the first occurrence's position and the last seen previous position,

5 variables are used to handle the special cases, three of them are for the cases when the number of replaceable occurrences increment differs from that of the decrement and two of them are for the both sides of the first occurrence of the pair,
when all the occurrences of a pair have been studied then these two arrays' modified positions are traversed, the number of replaceable occurrences changes are applied from the 5 variables and from the two arrays to records which are then moved into proper places in priority queue, the new pairs' records are created and the information is copied to them and they are inserted into proper positions in the priority queue.

[c4] A method of decompressing a compressed representation of a sequence of symbols, said compressed representation comprising of atomic symbols i.e. symbols from original uncompressed input and pointers to positions in compressed sequence, a pointer being a value of the position pointed to plus a value greater than any atomic symbol's value, said decompressing method comprising the steps of:

For each position X in compressed input:

(a) determining if a position X in compressed input contains an atomic symbol or a pointer

(b) if a position X contains an atomic symbol then

copying the atomic symbol to next unoccupied position Z in decompressed output and setting

Length_of_position (X) = 1 and Starting_position(X) = Z

else if a position X contains a pointer to a position Y then setting

E=Length_of_position(Y) + Length_of_position(Y+1)

and copying E positions to next unoccupied position Z in decompressed output from decompressed output from the position Starting_position(Y) and setting

Length_of_position(X) = E and Starting_position(X) = Z.

[c5] A method of decompressing a compressed representation of a sequence of

symbols, said compressed representation comprising of atomic symbols i.e. symbols from original uncompressed input and pointers to positions in compressed sequence, a pointer being a value of the position pointed to plus a value greater than any atomic symbol's value, said decompressing method comprising the steps of:

(a) traversing the compressed positions once and for each position Y pointed to

if IspointedTo(Y) is empty then

j = j + 1

IsPointedTo(Y) = j

if IspointedTo(Y+1) is empty then

j = j + 1

IsPointedTo(Y+1) = j

(b) traversing the compressed positions once and for each position X in compressed input:

(1) determining if a position X in compressed input contains an atomic symbol or a pointer

(2) If a position X contains an atomic symbol then

copying the atomic symbol to the decompressed output to next unoccupied position Z

If IsPointedTo(X) is not empty then

W= IsPointedTo(X)

Length_of_position(W) = 1

Starting_position(W) = Z

else if a position X contains a pointer to a position Y then:

W1 = IsPointedTo(Y)

W2 = IsPointedTo(Y+1)

E=Length_of_position(W1) + Length_of_position(W2)

copying E positions to next unoccupied position Z in decompressed output from decompressed output from the position Starting_ position(W1) and setting

If IsPointedTo(X) is not empty then

W= IsPointedTo(X)

Length_of_position(W) = E

Starting_position(W) = Z.